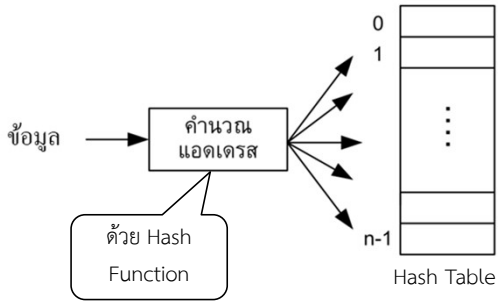
Hash,
Tree,
Binary Tree,
Binary Search Tree

การอบรมโอลิมปิกวิชาการและการพัฒนามาตรฐานวิทยาศาสตร์
และคณิตศาสตร์ สาขาคอมพิวเตอร์ ค่ายที่ 2
ศูนย์คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตปัตตานี

Hash

Hash



Hash Function

- ตัวอย่าง
  - การเก็บข้อมูลหมายเลขโทรศัพท์ของนักศึกษาในมหาวิทยาลัย (table[])
    - Index (ดัชนี) ที่ใช้อ้างอิงตำแหน่งของของข้อมูลในอาร์เรย์ คือ หมายเลขโทรศัพท์ ดังนั้นจะได้อาร์เรย์ table[telNo]
    - ตัวอย่าง หมายเลขโทรศัพท์ 123456789 เก็บข้อมูลในตำแหน่ง table[123456789]
    - ใช้เฉพาะข้อมูล 4 หลักสุดท้าย จะได้ table[6789]

5

Hash Function

- การเลือกหลัก (Selection Digits)
- การบวกหลัก (Folding)
- การหารเอาเศษ (Modulate arithmetic)
- การเปลี่ยนข้อความเป็นตัวเลข (Converting a character string to an integer)

6

Hash Function

- การเลือกหลัก (Selection Digits)
  - ตัวอย่าง
    - h(123456789) = 6789 (เลือกจากสี่หลักสุดท้าย)
    - h(123456789) = 19 (เลือกจากหลักที่หนึ่งและหลักสุดท้าย)
    - h(123456789) = 13579 (เลือกจากหลักที่เป็นเลขคี่)
  - ข้อควรระวัง ข้อมูลบางข้อมูลจะมีตัวเลขของหลักเดียวกันซ้ำกันเป็นจำนวนมาก

7

Hash Function

- การบวกหลัก (Folding)
  - การเลือกหลัก แล้วนำตัวเลขในหลักนั้นมาบวกกัน
  - ตัวอย่าง
    - h(123456789) = 6+7+8+9 = 30 (เลือกจากสี่หลักสุดท้าย)
    - h(123456789) = 1+9 = 10 (เลือกจากหลักที่หนึ่งและหลักสุดท้าย)
    - h(123456789) = 1+3+5+7+9 = 25 (เลือกจากหลักที่เป็นเลขคี่)
    - h(123456789) = 1+2+3+4+5+6+7+8+9 = 45

8

Hash Function

- การบวกหลัก (ต่อ)
  - ข้อสังเกต การบวกทุกหลักจากทั้งหมด 9 หลัก h(x) จะมีค่าตั้งแต่ 0-81
  - ถ้าต้องการเพิ่มขนาดของ Hash table ต้องปรับรูปแบบของการบวก เช่น
    - h(123456789) = 123+456+789 = 1368 ซึ่ง Hash Function นี้ทำให้ได้ขนาดของ hash table เท่ากับ 3*999 = 2997

9

Hash Function

- การหารเอาเศษ (Modulate arithmetic)
  - h(x) = x mod table_size
  - ตัวอย่าง กำหนดให้ hash table มีขนาดเท่ากับ 101
    - แปลงค่า 0 ≤ x ≤ 100
    - h(123456789) = 123456789 mod 101
      = 45
    - ข้อสังเกต จาก hash function นี้พบว่ามักจะมีข้อมูลแฮชในตำแหน่งที่ table[0] table[1] ซ้ำกัน

10

Hash Function

- การเปลี่ยนข้อความเป็นตัวเลข (Converting a character string to an integer)
  - ใช้รหัส ASCII
    - “NOTE” -> 78798469
  - ใช้ค่าตัวเลข เช่น 1-26 แทน A-Z
    - “NOTE” -> 1415205
  - ใช้ค่าตัวเลขแล้วเปลี่ยนให้เป็นเลขฐานสอง

11

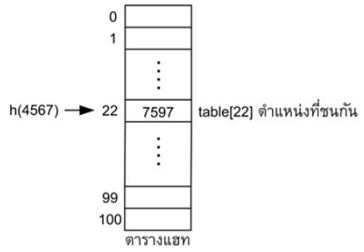
Hash Function

- การเปลี่ยนข้อความเป็นตัวเลข (ต่อ)
  - ใช้ค่าตัวเลขแล้วเปลี่ยนให้เป็นเลขฐานสอง
    - N = 14 = 01110_2
    - O = 15 = 01111_2
    - T = 20 = 10100_2
    - E = 5 = 00101_2
    - เมื่อเรียงต่อกันจะได้ 01110011111010000101 ซึ่งเท่ากับ 474,757

12

Collision

- การชนกันของคีย์ใน Hash Table
  - ตัวอย่าง กำหนดให้
    - h(7597) = 7597 mod 101 = 22
    - h(4567) = 4567 mod 101 = 22



13

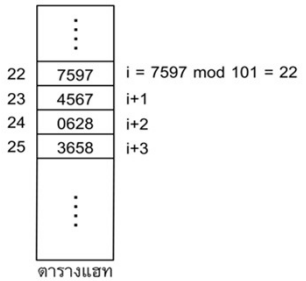
Resolving Collision

- ใช้ตำแหน่ง index ถัดไปที่ใกล้เคียงกัน
  - การแก้ปัญหาแบบลำดับ (Linear Probing)
  - การแก้ปัญหาแบบกำลังสอง (Quadratic Probing)
  - Double Hashing
- เปลี่ยนโครงสร้างของ Hash Table ให้สามารถเก็บข้อมูลได้มากกว่าหนึ่งข้อมูล
  - Separate Chaining

14

Resolving Collision

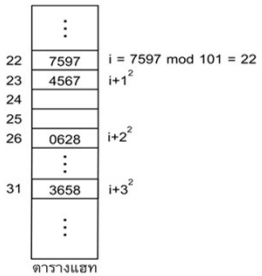
- การแก้ปัญหาแบบลำดับ (Linear Probing)
  - table[h(x)+1], table[h(x)+2], ...



15

Resolving Collision

- การแก้ปัญหาแบบกำลังสอง (Quadratic Probing)
  - table[h(x)+1^2], table[h(x)+2^2], table[h(x)+3^2], ...



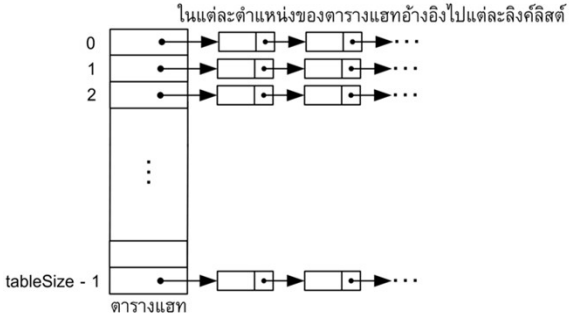
16

Resolving Collision

- Double Hashing
  - If table[$h_1(x)$] is not empty then use $h_2(x)$, $h_2(x) \neq 0$ and $h_1 \neq h_2$
  - ตัวอย่าง
    - $h_1(x) = x \bmod 11$
    - $h_2(x) = 7 - (x \bmod 7)$

Resolving Collision

- Separate Chaining



STL

- map
- unordered_map
- list
- set
- vector

ตัวอย่างที่ 2

- ศึกษาโปรแกรมตัวอย่าง (hash_chaining.cpp) จาก google classroom

---

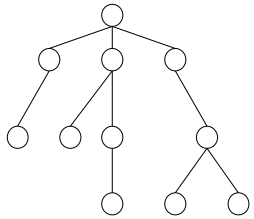
Problems (grader)

- Who Win
- Equal Adding

Tree

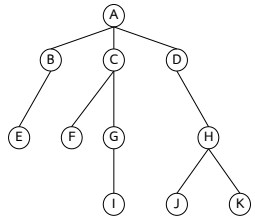
Tree

- โครงสร้างของทรี
  - เป็นโครงสร้างไม่เชิงเส้น (Non Linear)
  - เป็นโครงสร้างแบบลำดับชั้น (Hierarchical)
  - ความสัมพันธ์ของข้อมูล (ระหว่างโหนด) เป็นแบบแม่กับลูก (Parent-child)
  - ความสัมพันธ์ดังกล่าวถูกเชื่อมด้วยกิ่ง (หรืออาจเรียกว่า Edge หรือ Link)

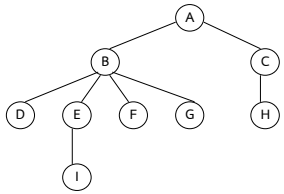


Tree

- โครงสร้างของทรี
  - โหนดราก (Root Node)
  - โหนดแม่ (Parent Node)
  - โหนดลูก (Child Node)
  - โหนดพี่น้อง (Sibling Node)
  - โหนดใบ (Leaf Node)
  - โหนดภายใน (Internal Node)
  - บรรพบุรุษ (Ancestor)
  - ทรีย่อย (Sub Tree)
  - ระดับของทรี (Level)



การแทนโครงสร้าง Tree ด้วยอาร์เรย์

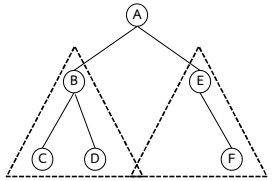


	0	1	2	3	4	5	6	7	8
Node	A	B	C	D	E	F	G	H	I
0: Child	1	3	7		8				
1: Child	2	4							
2: Child		5							
3: Child		6							

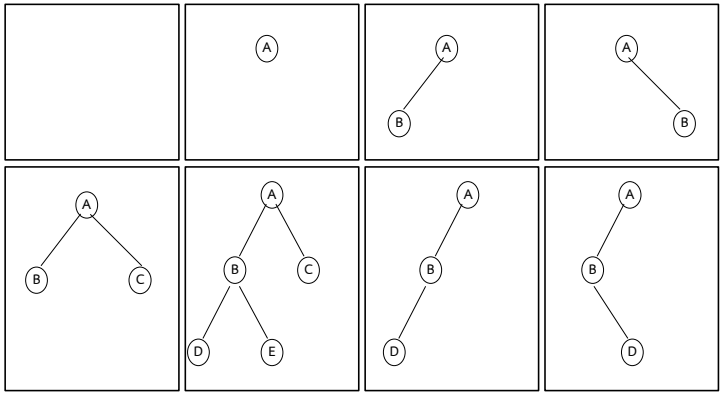
Binary Tree

Binary Tree

- โหนดแม่ (R) หนึ่งโหนดจะมีโหนดลูกได้ไม่เกิน 2 โหนด และเรียกโหนดลูกเหล่านั้นว่า ทรีย่อย (Subtrees)
- โหนดลูกที่อยู่ในตำแหน่งทางซ้ายของโหนดแม่จะเรียกว่า ทรีย่อยทางซ้าย (Left subtrees : T_L)
- โหนดลูกที่อยู่ในตำแหน่งทางขวาของโหนดแม่ เรียกว่า ทรีย่อยทางขวา (Right subtrees : T_R)

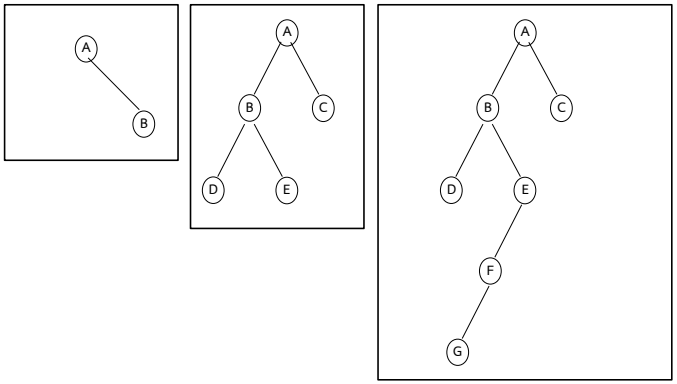


Binary Tree



คุณสมบัติของ Binary Tree

- ความสูงของทรี (Height of Trees)



คุณสมบัติของ Binary Tree

- ความสูงของทรี (Height of Trees) (ต่อ)
  - ความสูงที่มากที่สุด (ที่เป็นไปได้) ของไบนารีทรีที่มีโหนด N โหนด

    $H_{max} = N$
  - ความสูงที่น้อยที่สุดของของไบนารีทรีที่มีโหนด N โหนด

    $H_{min} = (\log_2 N) + 1$
  - จำนวนโหนดที่น้อยที่สุด (ที่เป็นไปได้) ของไบนารีทรีที่มีความสูง H

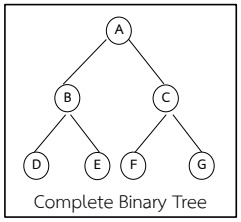
    $N_{min} = H$
  - จำนวนโหนดที่มากที่สุดของไบนารีทรีที่มีความสูง H

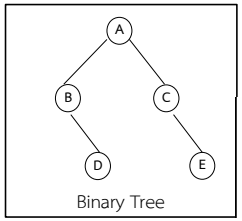
    $N_{max} = 2^H - 1$

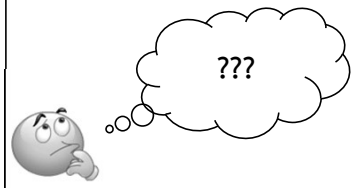
การแทนไบนารีทรีด้วยอาร์เรย์

- อาร์เรย์ 1 มิติ

A	B	C	D	E	F	G
0	1	2	3	4	5	6

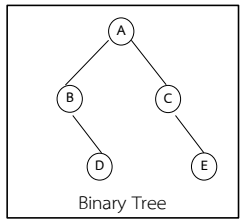


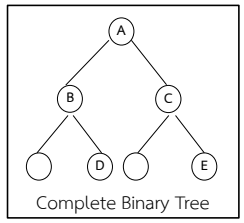




การแทนไบนารีทรีด้วยอาร์เรย์

- อาร์เรย์ 1 มิติ (ต่อ)





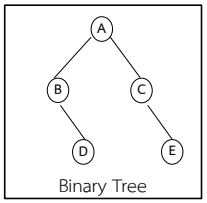
A	B	C		D		E
0	1	2	3	4	5	6

การแทนไบนารีทรีด้วยอาร์เรย์

- อาร์เรย์ 1 มิติ (ต่อ)
  - การหาตำแหน่งของโหนดแม่ (Parent Node: i)
    - กรณี i เป็นเลขคี่: ค่าสมบูรณ์ (Absolute Value) ของ i/2
    - กรณี i เป็นเลขคู่: ค่าสมบูรณ์ (Absolute Value) ของ i/2 ลบด้วย 1
  - การหาตำแหน่งของโหนดลูกทางซ้าย (Left child: i): 2i + 1
  - การหาตำแหน่งของโหนดลูกทางขวา (Right child: i): 2i + 2

การแทนไบนารีทรีด้วยอาร์เรย์

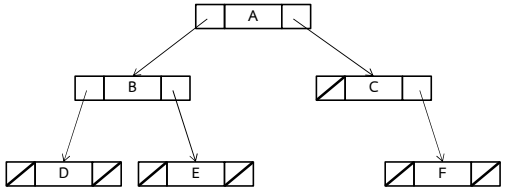
- อาร์เรย์ของ Struct



	Item	left	right
0	A	1	2
1	B		3
2	C		4
3	D		
4	E		

```
struct Node
{
   char item;
   int left;
   int right;
};
```

การแทนไบนารีทรีด้วยลิงค์ลิสต์



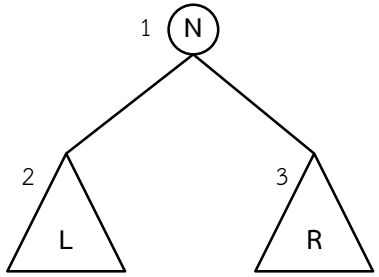

```
struct Node{
  int data;
  struct Node *left;
  struct Node *right;
};
```


การท่องเข้าไปในทรี (Binary Tree Traversals)

- วิธีการท่องแบบแนวลึก (Dept-First Traversals)
  - Preorder Traversal (NLR)
  - Inorder Traversal (LNR)
  - Postorder Traversal (LRN)
    - * N แทน Parent Node
      L แทน Left Sub-tree
      R แทน Right Sub-tree
- วิธีการท่องแบบแนวกว้าง (Breadth-First Traversal)

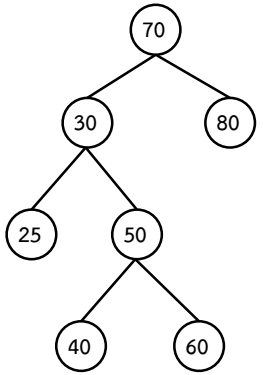
การท่องเข้าไปในทรี (Binary Tree Traversals)

- Preorder Traversal (NLR)



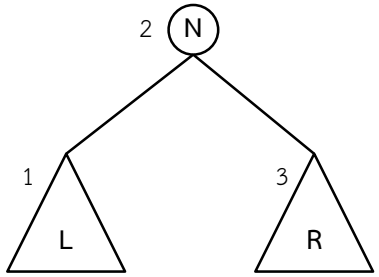
การท่องเข้าไปในทรี (Binary Tree Traversals)

- Preorder Traversal (NLR) (ต่อ)



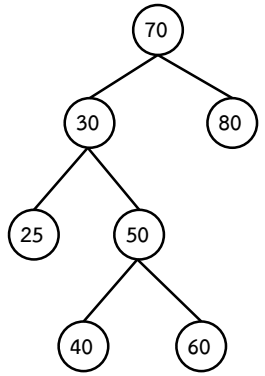
การท่องเข้าไปในทรี (Binary Tree Traversals)

- Inorder Traversal (LNR)



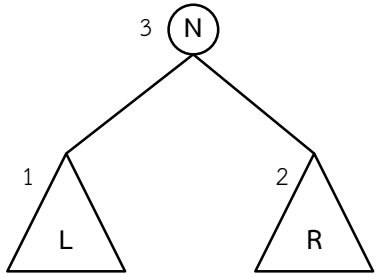
การท่องเข้าไปในทรี (Binary Tree Traversals)

- Inorder Traversal (LNR) (ต่อ)



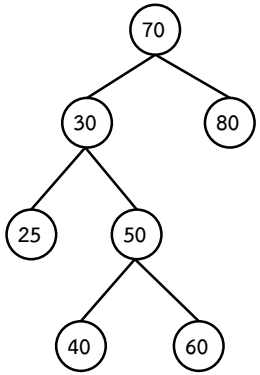
การท่องเข้าไปในทรี (Binary Tree Traversals)

- Postorder Traversal (LRN)



การท่องเข้าไปในทรี (Binary Tree Traversals)

- Postorder Traversal (LRN) (ต่อ)



41

ตัวอย่างที่ 1

- ศึกษาโปรแกรมตัวอย่าง (BT_Inorder.cpp) จาก google classroom ซึ่งเป็นการสร้าง Binary Tree จากอาร์เรย์ แล้วแสดงผลลัพธ์แบบ Inorder Traversal

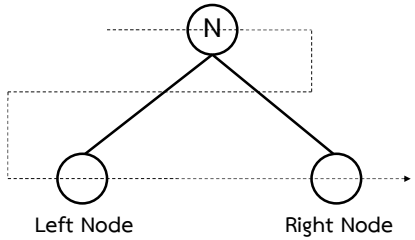
Problems (grader)

- binaryTree_1

42

การท่องเข้าไปในทรี (Binary Tree Traversals)

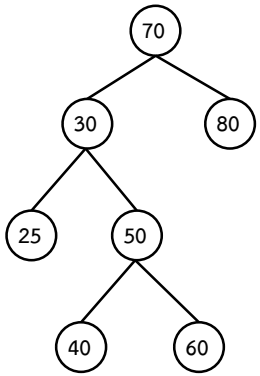
- วิธีการท่องแบบแนวกว้าง (Breadth-First Traversal)



43

การท่องเข้าไปในทรี (Binary Tree Traversals)

- วิธีการท่องแบบแนวกว้าง (ต่อ)



44

ตัวอย่างที่ 2

- ศึกษาโปรแกรมตัวอย่าง (BF_traversal.cpp) จาก google classroom

Problems (grader)

- BF_traversal_Q (Breadth-First Traversal using Queue)

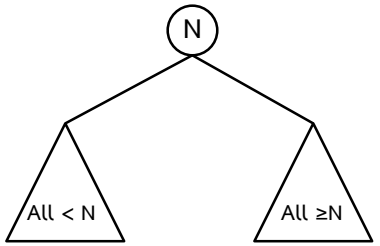
45

Binary Search Tree (BST)

46

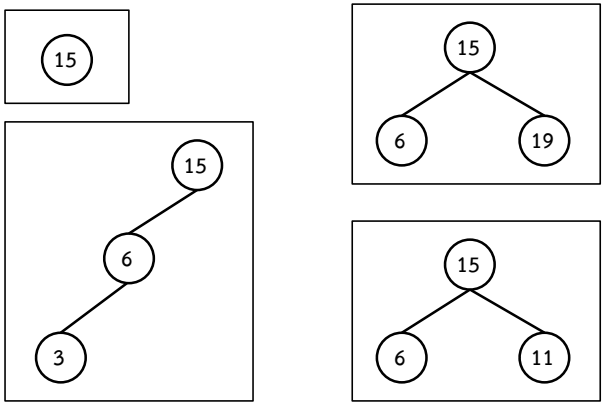
Binary Search Tree

- คุณสมบัติเบื้องต้น
  - ทุกโหนดของ Left Sub-tree ต้องมีค่าน้อยกว่า Parent Node
  - ทุกโหนดของ Right Sub-tree ต้องมีค่ามากกว่าหรือเท่ากับ Parent Node



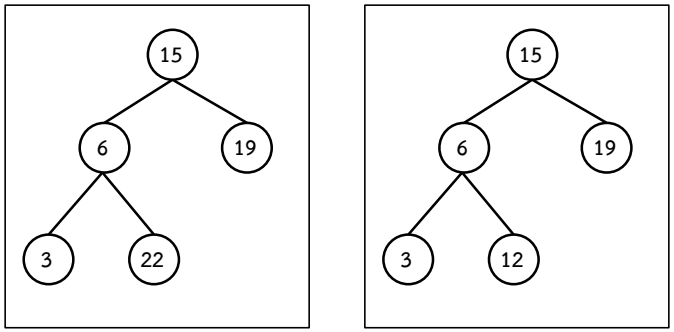
47

Binary Search Tree



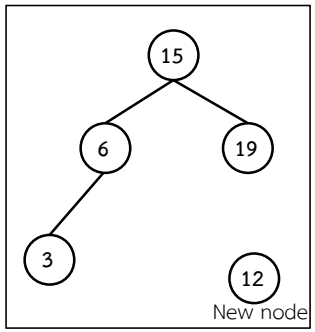
48

Binary Search Tree



Binary Search Tree Operations

- การแทรกโหนด (Insertion)



Binary Search Tree Operations

- การแทรกโหนด (Insertion)

```
1  InsertItem(bts: BinaryTree, newNode)
2     if (bts is NULL)
3        create a new node and set a reference
4     else if (newNode < bts's item)
5        InsertItem(left sub-tree of bts, newNode)
6     else
7        InsertItem(right sub-tree of bts, newNode)
```


Binary Search Tree

- การค้นหาข้อมูลในไบนารีทรี
  - การค้นหาข้อมูลที่มีค่าน้อยที่สุด

```
1  MinSearch(bts: BinaryTree)
2     if (left sub-tree of bts is empty)
3        return root
4     else
5        return MinSearch(left sub-tree of bts)
```


Binary Search Tree

- การค้นหาข้อมูลในไบนารีทรี
  - การค้นหาข้อมูลที่มีค่ามากที่สุด

```
1  MaxSearch(bts: BinaryTree)
2     if (right sub-tree of bts is empty)
3        return root
4     else
5        return MaxSearch(Right sub-tree of bts)
```

Binary Search Tree

- การค้นหาข้อมูลในไบนารีทรี

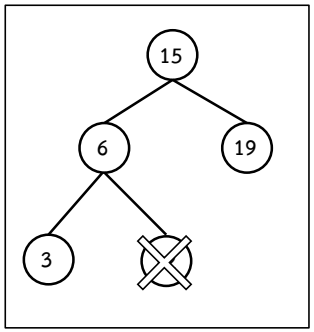
```
1  Search(bts: BinaryTree, key) //key: search key
2     if (bts is empty)
3        //not found
4     else if (bts's item == key)
5        // key is found
6     else if (key < bts's item)
7        search(left sub-tree of bts, key)
8     else
9        search(right sub-tree of bts, key)
```


Binary Search Tree Operations

- การลบโหนด (Deletion)
  - การลบโหนดใบ
  - การลบโหนดภายในที่มีโหนดลูก 1 โหนด
  - การลบโหนดภายในที่มีโหนดลูก 2 โหนด

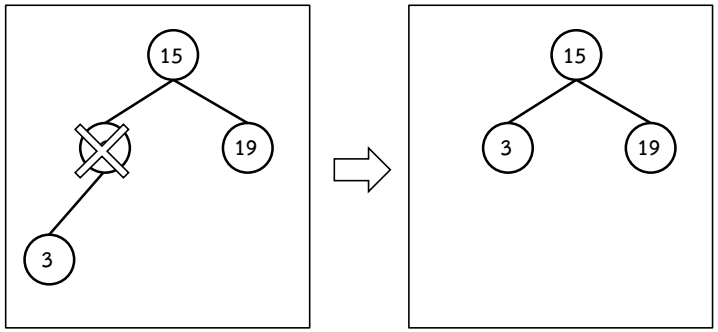
Binary Search Tree Operations

- การลบโหนดใบ



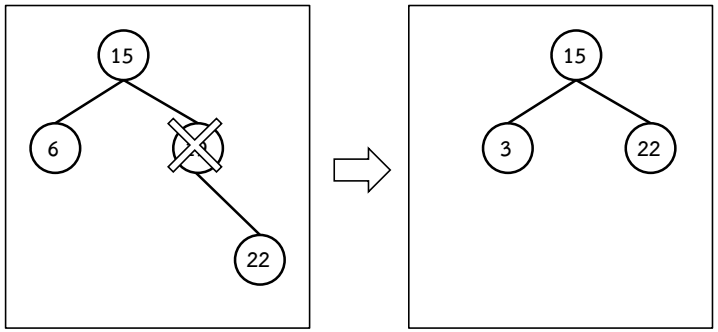
Binary Search Tree Operations

- การลบโหนดภายในที่มีโหนดลูก 1 โหนด



Binary Search Tree Operations

- การลบโหนดภายในที่มีโหนดลูก 1 โหนด

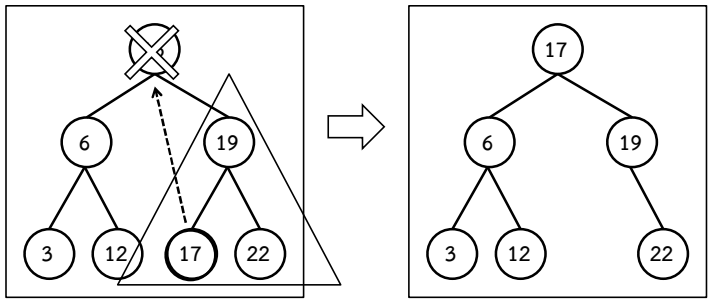


Binary Search Tree Operations

- การลบโหนดภายในที่มีโหนดลูก 2 โหนด
  - Inorder Successor: เลือกโหนดใบในตำแหน่งซ้ายสุดของ Right Sub-tree มาแทนที่ Root Node
  - Preorder Successor: เลือกโหนดใบในตำแหน่งขวาสุดของ Left Sub-tree มาแทนที่ Root Node

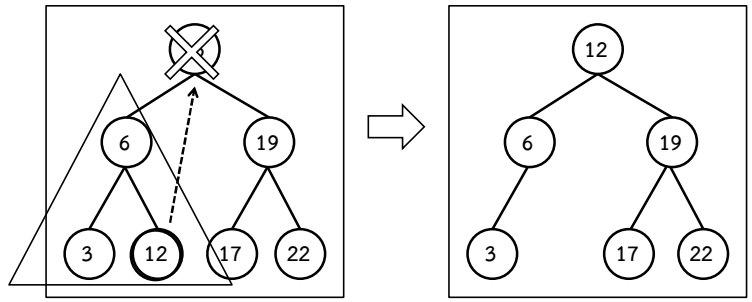
Binary Search Tree Operations

- การลบโหนดภายในที่มีโหนดลูก 2 โหนด
  - Inorder Successor



Binary Search Tree Operations

- การลบโหนดภายในที่มีโหนดลูก 2 โหนด
  - Preorder Successor



Problems (grader)

- BST_1
- Leave Node
- Expression Tree
- Maximum Path Sum of Binary Tree